Early Containment of Fast Network Worm Malware

Muhammad Aminu Ahmad*, Steve Woodhead* and Diane Gan**
Internet Security Research Laboratory
*Department of Engineering Science
**Department of Computing and Information System
University of Greenwich, UK
Email:{m.ahmad, s.r.woodhead, d.gan}@gre.ac.uk

***Abstract*—This paper presents a countermeasure mechanism for the propagation of fast network worm malware. The mechanism uses a cross layer architecture with a detection technique at the network layer to identify worm infection and a data-link containment solution to block an identified infected host. A software prototype of the mechanism has been used to demonstrate its effective. An empirical analysis of network worm propagation has been conducted to test the mechanism. The results show that the developed mechanism is effective in containing self-propagating malware with almost no false positives.**

Keywords: Containment, worm detection, malware, cyber defence

I. Introduction

The Internet has provided a medium for communication and sharing of information amongst people, businesses, governments and organisations. Therefore the Internet must be kept continuous and secured from any form of malicious activities such as unauthorised access to computer system and malware attacks. Malicious software (malware) is a generic term for any software that enters a computer system without the authorisation of the user to perform unwanted actions [1]. Malware can be classified under a number of headings, including viruses, worms, trojans, spyware, adware, rootkits, drive-by downloads and other malicious and unwanted software. Self-propagating malware, termed a worm, is a malicious software program that propagates across a network by infecting hosts and in some cases launching malicious activities. A scanning network worm propagates by probing pseudo-random addresses looking for vulnerable hosts, which makes the malware highly virulent in nature. Fast scanning network worms are a particularly dangerous sub-class of such software.

The Internet has experienced a number of notable worm outbreaks that caused disruption of services [2], damage to targeted systems [3], cyber espionage [4] and financial losses ranging from millions to billions of US Dollars [5]. Common countermeasures to worm infection are signature-based anti-virus software, network intrusion detection systems and host intrusion detection systems. However, the ability of such systems to counter the effects of fast scanning network worms is limited because their high propagation and infection rates pose a significant security threat, with consequent damage to networks and the Internet. Thus it is important to effectively identify and counter the propagation of worms, particularly fast scanning network worms, using a mechanism that must first work without the need to rely on signatures (sequence of byte-stream), because methods which rely on content signatures are unlikely to detect zero-day network worms. Additionally, due to the high propagation rates of fast scanning network worms, the traditional approach of waiting for patches to be released by vendors to fix vulnerabilities is not viable.

This paper presents an advance cross-layer detection and containment technique as an improvement over the NEDAC mechanism reported in [2]. The NEDAC mechanism uses datagram-header information at the network layer to detect the presence of fast scanning worms and a containment solution at data-link layer to block outgoing traffic from a host that has been identified as infected. The rest of the paper is organised as follows. Section 2 summarises related work on worm detection and containment systems. Section 3 presents a description of the NEDAC mechanism. Section 4 presents the evaluation method used to test the NEDAC mechanism. Section 5 presents the experiments conducted and Section 6 presents a discussion of the experimental results obtained. Section 7 concludes this paper and points out possible future work.

II. Related Work

A range of anomaly-based network intrusion detection systems have been developed to identify the presence of worms using datagram header information and payload information [6].

Gu et al. [7] developed an algorithm, termed DSC, that correlates incoming and outgoing traffic, i.e., if a host received a datagram on port i, and then starts sending datagrams destined for port i, it becomes a suspect. Jung et al. [8] proposed an algorithm, termed TRW, which identifies a remote host attempt to establish a new TCP connection to a local destination as normal if there is a corresponding TCP reply. On the other hand, failure to establish a successful TCP connection is considered suspicious. Weaver et al. [9] simplified the TRW scheme by considering all new connections to be a failure until a response is received. The algorithm drops a datagram if it does not match an existing and successfully-established connection after a predefined threshold count. Whyte et al. [10] used DNS-based rate limiting to suppress scanning worms in an enterprise network by identifying the absence of DNS resolution before a new connection as anomalous. Shahzad

and Woodhead [11] proposed a scheme that uses the absence of DNS lookup action prior to an outgoing TCP SYN or UDP datagram to a new destination IP address to detect worm propagation, and a protocol termed Friends to spread reports of an identified worm event to potentially vulnerable and uninfected peer networks within the scheme. Li and Stafford [12] proposed a worm detector, which they termed SWORD. SWORD comprises two main modules; a Burst Duration Detector (BDD) and a Quiescent Period Detector (QPD). The BDD module encompasses a burst detection algorithm to prevent fast scanning worms by creating a window for every different size of first-contact connections. The QPD module ensures that quiescent periods in network activity do not disappear because of constant worm scanning. These techniques consume resources in order to keep track of distinct connection and host information, especially in large networks [13], and they can only slow worm infections [2].

Additionally, Wang and Stolfo [14] and Kim et al. [15] proposed payload-based anomaly detection schemes. Wang and Stolfo [14] proposed a detection scheme known as PAYL to detect and generate signatures for zero-day worms. PAYL uses a training phase to create a profile during normal operation, and produces a byte frequency distribution as a model for normal payloads. Based on this information, a centroid model is created and then during the detection phase, the Mahalanobis distance of each datagram payload from the centroid model is calculated. A datagram is considered to be anomalous based on its distance from the normal behaviour. Kim et al. [15] proposed a detection scheme using a standalone device. The scheme employed the detection method reported by Kim et al. in [16]. During the training phase, the mean and standard deviation scores for all datagrams are computed. In the detection phase, a score is computed by counting the number of datagram bytes that fall outside the range defined for each byte. These mechanisms have limitations such as computational complexity [17], management overhead [18], high rates of false positives [13] and incur significant delays in deployment and detection [15].

III. The NEDAC Mechanism

The NEDAC mechanism comprises a network layer detection system and a containment system at the data-link layer that work together to provide a countermeasure solution, with a connection maintained between the two systems to enable continuous data transmission. The detection system detects anomalies from client hosts and server hosts in a network using different techniques. The detection system maintains a list of server IP addresses in order to differentiate client and server hosts in a network. Client hosts are defined as network hosts which typically consume Internet services (e.g. workstations, laptops, tablets, smart-phones, etc) while server hosts are network hosts used to serve client requests (e.g. web servers and email servers). The detection system keeps track of inbound and outbound TCP SYN and UDP datagrams for a window of time with value T to determine anomalies that exceed a threshold. The threshold is a maximum allowable count of anomalous datagrams a host can send before T has elapsed. The containment system receives the MAC address of an identified infected host from the detection system and then blocks all traffic originating from the host using MAC address access control. The working mechanism of NEDAC is presented in Algorithm 1.

Algorithm 1 The NEDAC detection and containment system

```
Begin
/* Initialize tables */
initializeTable(inboundTable)
initilaizeTable(resolutionTable)
initilaizeTable(noResolutionTable)
/* initialize exempt table*/
initializeTable(exemptTable)
/* Set timer */
T = SetTimerSignal()
/* open interface */
openInterface(interface)
 /* do in parallel */ process 1
while (there are datagrams to process) do
    getDatagram()
    uniqueMAC = getMACAddress(datagram)
    headerInfo = getHeaderInfo(srcIP,dstIP,sPort,dPort)
    if datagram is inbound then
        if packet is DNS reply then
            updateResolutionTable()
        else
            updateInboundTable(dstIP)
            updateInboundTable(dPort)
        end if
    else
        if headerInfo is not found in exemptTable[] then
            if source host not a server then
                if IP addresses not in resolutionTable then
                    updateNoResolutionTable(headerInfo)
                    if (dPortCounter > threshold) then
                        containHost(MAC)
                        if dPort in inboundTable then
                            blockInbound(dPort)
                        end if
                    end if
                else
                    getNextDatagram()
                end if
            else
                correlate(headerInfo)
                if dport in inboundTable then
                    updateInboundTable(headerInfo)
                end if
                if (dPortCounter > threshold) then
                    containHost(MAC)
                    if dport in inboundTable then
                        blockInbound(dPort)
                    end if
                end if
            end if
        else
            getNextDatagram()
        end if
    end if
end while
/* do in parallel */ process 2
while true do
    if (T generates timeout signal) then
        for entries in resolutionTable do
            if (TTL >= 86400) then
                delete entry
            end if
        end for
        for entries in noResolutionTable and inboundTable do
            if (TTL >= 60) then
                delete entry
            end if
            halveThresholds()
        end for
    end if
end while
End
```

The NEDAC algorithm monitors TCP SYN and UDP data-

grams from hosts in a network. For client hosts, the algorithm observes DNS resolution datagrams and records the IP address of the host that made the resolution and the resolved address in the resolution table. The destination IP address and port of an inbound datagram (excluding a DNS reply) is recorded in the inbound table for both client and server hosts. Outgoing datagram header information (source IP addresses and ports) is associated to entries in an exempt table. The exempt table comprises a list of IP addresses and ports that are exempt from the algorithm. If the header information results in a miss, the algorithm determines whether there is a recent DNS query by a client host for the destination IP address prior to sending the datagram by checking the resolution cache. If there is a miss, the algorithm records the destination port in the no-resolution cache, increments its counter and then determines excess using the threshold with value V. For server hosts, the algorithm checks the presence of the destination port of outbound TCP SYN and UDP datagrams in the inbound table. If there is a hit, a counter for such entry is incremented for TCP datagrams. An additional verification of the destination IP address is made to determine a reply UDP datagram, and if the destination IP address does not match the IP address recorded for such entry in the inbound table, its counter is incremented and then excess in threshold is also determined. Upon a host exceeding the set threshold, the algorithm invokes the containment system and then checks the presence of the suspect port in the inbound cache. If there is a hit, an additional countermeasure is applied at the network layer using an access control list (ACL) to block all inbound datagrams destined for the suspect port in the network segment. A time-to-live (TTL) is provided for entries in all the caches. The default TTL value for DNS (86400 seconds) is applied to the resolution cache and 60 seconds is applied to the no-resolution and inbound caches. Furthermore, the algorithm decrements the counters in the no-resolution and inbound tables by half after the expiration of a timing window of T, and then checks all caches to determine and remove entries with expired TTL values.

The improvements of the countermeasure mechanism on the previous technique in [2] are (1) separate detection techniques for client and server hosts to improve effectiveness of the system (2) an additional countermeasure mechanism for inbound worm traffic to block remote to local worm infection and (3) time-to-live for records maintained in caches to reduce excessive resource consumption.

IV. Evaluation Procedure

To evaluate the proposed mechanism, a software prototype was developed and tested using worm propagation experiments in a controlled environment. The NEDAC mechanism was tested along with two previously reported worm detection techniques namely DSC and DNS-based detection schemes. The schemes were also implemented in software based on the description provided by their authors. The DNS-based scheme was termed DNS-RL.

The testing environment used for the evaluation process is a virtualised testbed described in [19]. The testbed contains four virtualised enterprise networks comprising a number of virtual network cells. The testbed has a scale of 1200 virtual machines, supports the use of worm daemons and has utilities for replaying network traces as background traffic. To generate background traffic during the worm propagation experiments, the evaluation used the DARPA 1999 evaluation dataset [20]. The "inside" traces of weeks 1 and 3 of the dataset meet the requirements of the evaluation because they are attack free traces that contain payload information for the variety of protocols needed. Additionally, the traces include a wide range of collected traffic from 31 network hosts.

The evaluation process used two contemporary pseudo-worms that were developed based on the Microsoft RDP (CVE-2012-0002) vulnerability [21] of 2012 and the Shell-Shock vulnerability [22] of 2014. Ahmad and Woodhead [2] reported the likely susceptible population values and potential datagram sizes of the Microsoft RDP and ShellShock vulnerabilities as circa 16.5M and 3800 bytes and 42.5k and 2000 bytes respectively.

Additionally, the bandwidth available for an infected host and the worm datagram size determine how fast a worm can send datagrams. The average Internet connection speed was estimated to be within the range 10 Mbps to 1000 Mbps [23]. Although it is impossible for a host to achieve the maximum speed of a network card, the vast majority of Internet connected hosts are capable of transmitting data at 60 Mbps to 120 Mbps [24]. Thus based on the assumption that the Internet connected hosts exhibit an average data transmission rate of 90 Mbps, the scan rate S, required for a single worm instance to transmit a datagram of size M (in bytes), over a C megabits Internet connection per second can be determined using $S = \frac{C}{(M*8)}$. Therefore, the likely scan rates for the RDP and ShellShock pseudo-worms are $\left(\frac{90{,}000{,}000}{3800*8}\right) = 2960$ and $\left(\frac{90{,}000{,}000}{2000*8}\right) = 5625$ datagrams per second respectively.

The scan rates of the pseudo-worms were scaled down by a factor of 24 and 45 for the RDP and ShellShock pseudo-worms respectively to avoid overloading server resources. The resulting scan rates employed in the experiments are 125 "infectious" datagrams per second for RDP and ShellShock. Furthermore, the results of the experiments were scaled up by a factor of 24 and 45 for the RDP and ShellShock pseudo-worms respectively.

Ahmad and Woodhead [2] reported the number of susceptible hosts per million Internet hosts for RDP and ShellShock pseudo-worms as 4454 and 12 respectively. Thus, due to the scale of the testbed used, which has a maximum number of 1200 hosts, four class B size$\left(2^{16}\right)$ networks were used for RDP and five class A size $\left(2^{24}\right)$ networks were used for Shell-Shock. Thus, the resulting values were $\left[\left(2^{16}\right) * 4 * \left(\frac{4454}{1000000}\right)\right]$ = 1168 and $\left[\left(2^{24}\right) * 5 * \left(\frac{12}{1000000}\right)\right] = 1007$ susceptible hosts respectively, within the relevant network address space.

V. Experimentation Setup

The evaluation experiments were conducted using the software prototypes of NEDAC DSC and DNS-RL. During the evaluation, a prototype of a detection scheme was positioned on the gateways of each network, and for NEDAC, the containment system was positioned on the switches as depicted in Fig.

1. RDP and ShellShock worm propagation experiments were conducted using random and then hit-list scanning behaviours for each detection scheme. The random scanning technique probes IPv4 addresses within the routable address space. The hit-list scanning technique infects a list of pre-compiled vulnerable hosts and then each infected host uses random scanning.

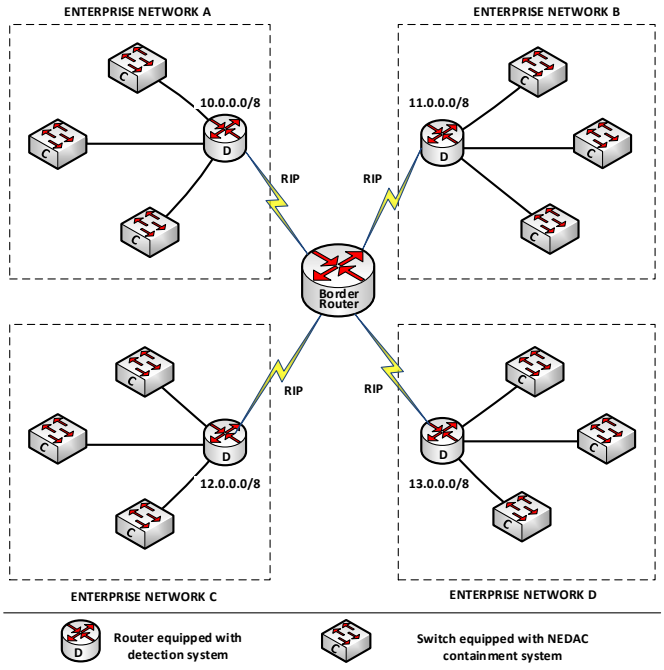

Fig. 1: Prototype setup

For each pseudo-worm experiment, a number of hosts (1168 for RDP and 1007 for ShellShock) were configured with the correct daemon to make them vulnerable to worm attack datagrams while other hosts were configured to replay the DARPA traces as background traffic. The worm attack and traffic replay events were executed concurrently in each experiment. The experiments were conducted without any countermeasures in place, then repeated with the countermeasures and the DARPA dataset as background traffic using threshold values of 100 through 400 anomalous datagrams sent by a host in a timing window of 10 seconds. The worm infection event was initiated by sending a UDP datagram to one of the vulnerable hosts.

A. RDP Pseudo-worm

The RDP pseudo-worm experiment was conducted using 1160 client hosts and 8 server hosts. The pseudo-worm daemon was configured to listen on UDP port 3389 and then transmit UDP datagrams to port 3389 at a scan rate of 125 "infectious" datagrams per second, once "infected". Five RDP pseudo-worm experiments were conducted using one initially infected host. Fig. 2 shows the average result of the five experiments.

The RDP-based worm experiment was repeated with a hit-list [25] of 10 and 20 hosts. Fig. 3 and Fig. 4 show the results of RDP pseudo-worm propagation using hit-lists of 10 and 20 hosts.

B. ShellShock Pseudo-worm

The ShellShock pseudo-worm experiment was conducted using 996 client hosts and 10 server hosts. The pseudo-worm daemon was configured to listen on UDP port 8080 and then transmit UDP datagrams to port 8080 at a scan rate of 125 "infectious" datagrams per second, once "infected". Five ShellShock pseudo-worm experiments were conducted using one initially infected host. Fig. 5 shows the average result of the five experiments.

As with RDP, the ShellShock worm experiment was repeated with a hit-list [25] of 10 and 20 hosts. Fig. 6 and Fig. 7 show the results of RDP pseudo-worm propagation using hit-lists of 10 and 20 hosts.

VI. Discussion

The section discusses the infection behaviours of the candidate pseudo-worms using random and hit-list scanning and the false positives observed during the experiments.

A. Random Scanning Infection

The results of random infection behaviours for the RDP and ShellShock pseudo-worms using a threshold value of 100 are presented in Fig. 2 and Fig. 5. When no countermeasure solution was in place, the RDP pseudo-worm infected 95% (1110) of the hosts in eight seconds as shown in Fig. 2. Additionally, the ShellShock pseudo-worm infected 95% (956) of its susceptible hosts in 145 seconds as shown in Fig. 5. When the detection schemes were applied, the infections were delayed and suppressed by DSC and DNS-RL and blocked completely by NEDAC. With DSC and DNS-RL, the RDP pseudo-worm infection was delayed by 12 seconds and suppressed to 44% (510) and 50% (580) respectively. The worm infections were detected by the DSC and DNS-RL schemes and the countermeasure solution was applied, but the initially infected host continued sending infectious datagrams, which infected a large number of hosts. However, with NEDAC, the initially infected host, for each pseudo-worm experiment, was detected and then blocked from sending out datagrams at the data-link layer, which stopped the infection completely for each of the two worm outbreak scenarios. Additionally, the NEDAC scheme blocked inbound traffic destined for the destination port used by the identified worm infection at the network layer, which also enable the mechanism to contain the worm infection quickly.

B. Hit-list Scanning Infection

Fig. 3 and Fig. 6 show the results of the worm experiments conducted with a hit-list of 10 hosts. When no countermeasure was in place, the RDP pseudo-worm infected 95% (1110) of the hosts in 6 seconds as shown in Fig. 3. The ShellShock pseudo-worm attained 95% (956) infection in 55 seconds as shown in Fig. 6.

With the DSC and DNS-RL scheme, the RDP pseudo-worm infection attained 95% in 19 and 15 seconds respectively. The ShellShock pseudo-worm attained 95% infection

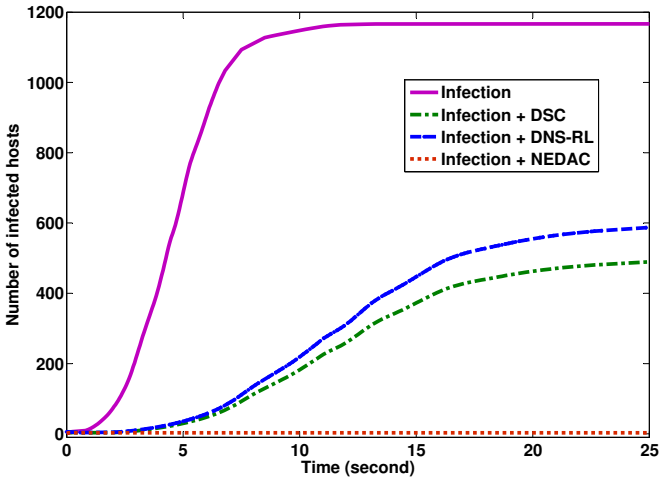

Fig. 2: RDP random scanning

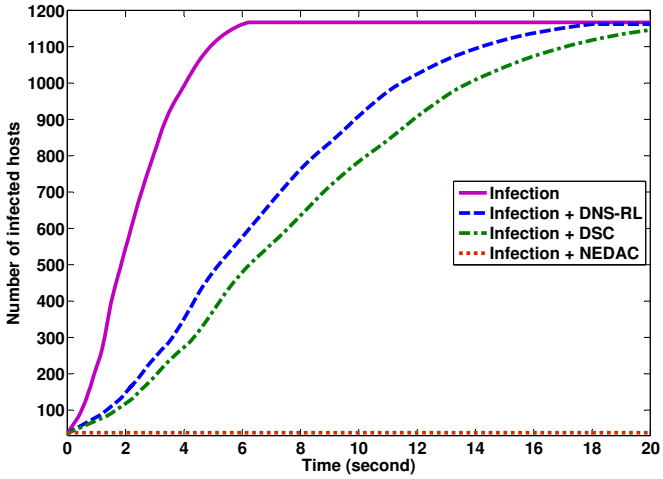

Fig. 3: RDP with a hit-list of 10 hosts

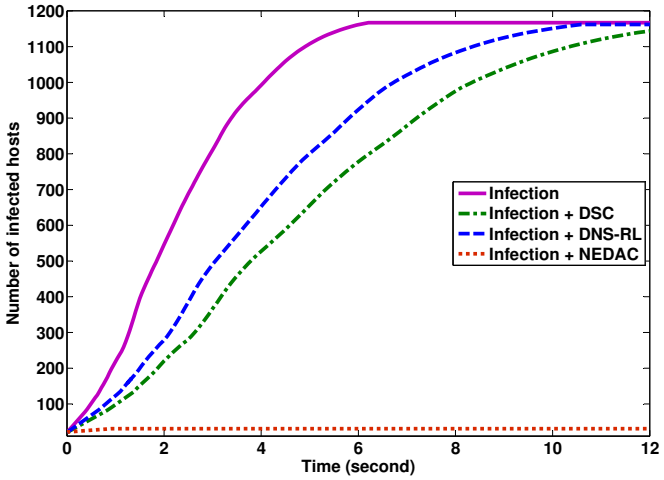

Fig. 4: RDP with a hit-list of 20 hosts

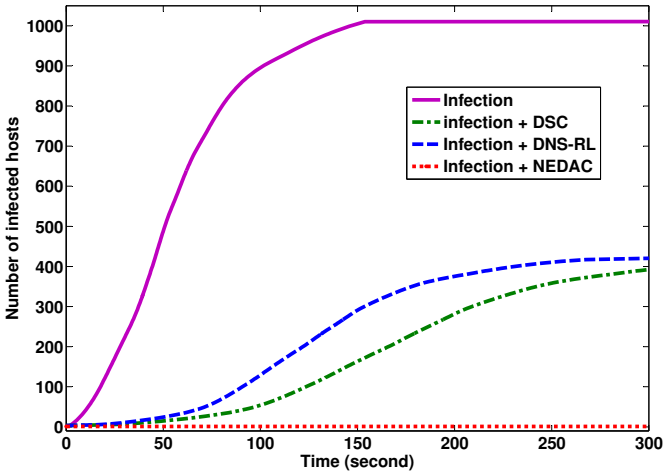

Fig. 5: ShellShock random scanning

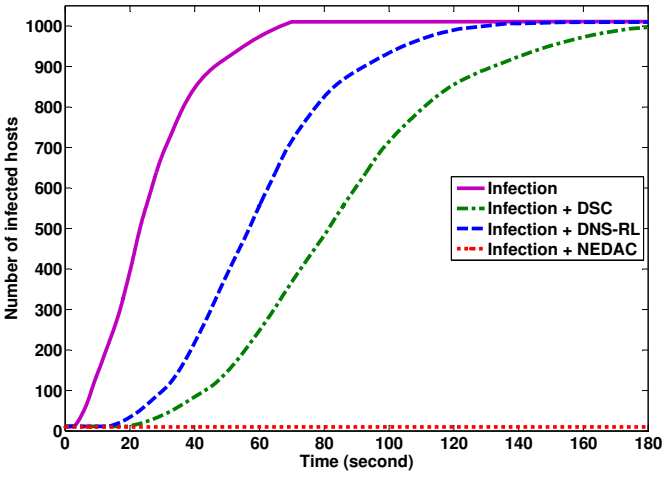

Fig. 6: ShellShock with a hit-list of 10 hosts

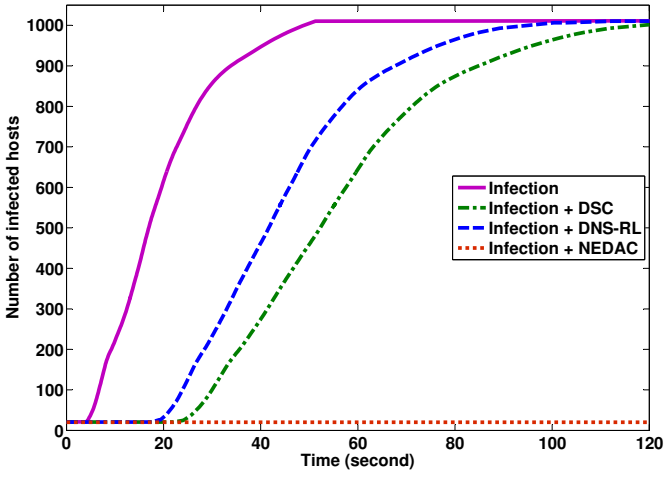

Fig. 7: ShellShock with a hit-list of 20 hosts

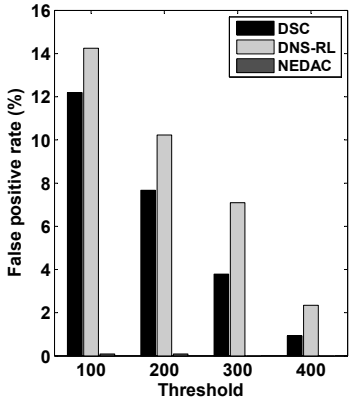

Fig. 8: False positive rate for RDP experiment

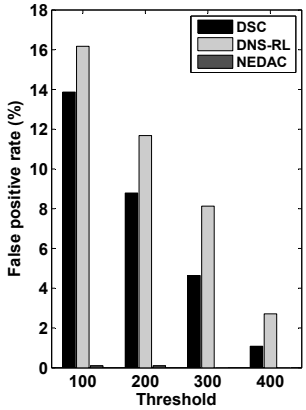

Fig. 9: False positive rate for ShellShock experiment

in 150 and 100 seconds with DSC and DNS-RL respectively. Furthermore, nine further infections were observed with NEDAC during the RDP pseudo-worm propagation and no further infections were observed during the propagation of the ShellShock pseudo-worm.

Fig. 4 and Fig. 7 show the results of the worm experiments conducted with a hit-list of 20 hosts. When no countermeasure was in place, the RDP pseudo-worm infected 95% (1004) of the hosts in 5 seconds as shown in Fig. 4. The ShellShock pseudo-worm attained 95% infection in 40 seconds as shown in Fig. 7. Furthermore, with the DSC and DNS-RL scheme, the RDP pseudo-worm infection attained 95% in 11 and 9 seconds respectively. The ShellShock pseudo-worm attained 95% infection in 90 and 75 seconds with DSC and DNS-RL respectively. For NEDAC, 56 further infections were observed during the RDP pseudo-worm propagation and no further infections were observed during the propagation of the ShellShock pseudo-worm.

C. Detection Performance

The false positive rates observed by the three detection schemes are presented in Fig. 8 and Fig. 9 for the RDP and ShellShock pseudo-worm experiments respectively. The schemes detected all real pseudo-worm datagrams in all the experiments conducted and therefore the true positive (TP) rates are 100%. However, the DSC and DNS-RL schemes incurred higher rates of false positives (FP) than NEDAC. NEDAC has very low FP rates using 100 and 200 as thresholds and zero FP rates using 300 and 400 as thresholds. Generally, the false positive rate diminishes with rising threshold values. NEDAC raised one false positive with threshold values of 100 and 200, which was caused by a multicast UDP datagram sent by a host to port 520, i.e, a RIP advertisement. Additionally, the rate at which RIP sends updates to neighbouring routers is not similar to fast scanning worm behaviour because RIP routers exchange update every 30 seconds by default. Nevertheless, the RIP port can be added into the exempt list in NEDAC to avoid false positives. Across the whole experimental data set, NEDAC has a better performance in terms of false positives compared to the DNS-RL scheme.

VII. Conclusion and Future work

This paper has presented a countermeasure solution against fast scanning network worms. A software prototype of the worm countermeasure solution was used to evaluate the scheme using a set of experiments. The results of the experiments showed that the countermeasure solution is sensitive in detecting and containing an identified worm infection with almost no false positives. The results of a comparative analysis showed that the countermeasure solution has a better performance compared to two previously reported detection schemes.

As for future work, it is desirable to evaluate the mechanism using different background traffic. The aim of improving the detection system to use dynamic threshold policy and the speed of containment will also be investigated. Furthermore, the effect of timing window size and volume of background traffic will be investigated.

References

[1] Jarno Niemelä and Pirkka Palomäki. Malware detection by application monitoring, November 19 2013. US Patent 8,590,045.

[2] Muhammad Aminu Ahmad and Steve Woodhead. Containment of fast scanning computer network worms. In *Internet and Distributed Computing Systems*, volume 9258 of *Lecture Notes in Computer Science*, pages 235–247. Springer International Publishing, 2015.

[3] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5, 2011.

[4] Boldizsár Bencsáth, Gábor Pék, Levente Buttyán, and Mark Felegyhazi. The cousins of stuxnet: Duqu, flame, and gauss. *Future Internet*, 4(4):971–1003, 2012.

[5] Craig Fosnock. Computer worms: past, present, and future. *East Carolina University*, 8, 2005.

[6] Faisal M Cheema, Adeel Akram, and Zeshan Iqbal. Comparative evaluation of header vs. payload based network anomaly detectors. In *Proceedings of the World congress on Engineering*, volume 1, pages 1–5, 2009.

[7] Guofei Gu, Monirul Sharif, Xinzhou Qin, David Dagon, Wenke Lee, and George Riley. Worm detection, early warning and response based on local victim information. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 136–145. IEEE, 2004.

[8] Jaeyeon Jung, Vern Paxson, Arthur W Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 211–225. IEEE, 2004.

[9] Nicholas Weaver, Stuart Staniford, and Vern Paxson. Very fast containment of scanning worms. In *USENIX Security Symposium*, volume 2, pages 16–85, 2004.

[10] David Whyte, Evangelos Kranakis, and Paul C van Oorschot. Dns-based detection of scanning worms in an enterprise network. In *NDSS*, 2005.

[11] Khurram Shahzad and Steve Woodhead. Towards automated distributed containment of zero-day network worms. In *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, pages 1–7. IEEE, 2014.

[12] Jun Li and Shad Stafford. Detecting smart, self-propagating internet worms. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 193–201. IEEE, 2014.

[13] Pele Li, Mehdi Salour, and Xiao Su. A survey of internet worm detection and containment. *Communications Surveys & Tutorials, IEEE*, 10(1):20–35, 2008.

[14] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[15] Sun-il Kim, Nnamdi Nwanze, William Edmonds, Blake Johnson, and Paloma Field. On network intrusion detection for deployment in the wild. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 253–260. IEEE, 2012.

[16] Sun-il Kim and Nnamdi Nwanze. Noise-resistant payload anomaly detection for network intrusion detection systems. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 517–523. IEEE, 2008.

[17] V Jyothsna, VV Rama Prasad, and K Munivara Prasad. A review of anomaly based intrusion detection systems. *International Journal of Computer Applications*, 28(7):26–35, 2011.

[18] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[19] Muhammad Aminu Ahmad, Steve Woodhead, and Diane Gan. The v-network testbed for malware analysis. In *International Conference on Advanced Communication Control & Computing (ICACCCT)*, May 2016.

[20] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[21] CVE. *Common Vulnerabilities and Exposures*. [Online]. Accessed on 19th October 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0002.

[22] CVE. *Common Vulnerabilities and Exposures*. [Online]. Accessed on 19th October 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271.

[23] *Net Index*. [Online]. Accessed 16 November 2014. Available: http://www.netindex.com/.

[24] Marshini Chetty, David Haslem, Andrew Baird, Ugochi Ofoha, Bethany Sumner, and Rebecca Grinter. Why is my internet slow?: making network speeds visible. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1889–1898. ACM, 2011.

[25] Stuart Staniford, Vern Paxson, Nicholas Weaver, et al. How to own the internet in your spare time. In *USENIX Security Symposium*, pages 149–167, 2002.